# Introduction to Interactive Ray-Tracing

Philipp Slusallek

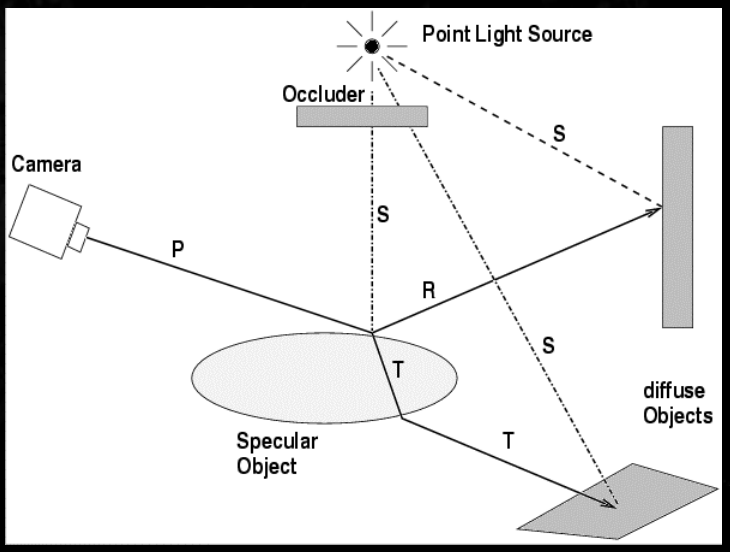
SIGGRAPH 2001

## Overview

- **Motivation**
- **Ray-Tracing Algorithm**
  - Ray generation, traversal, intersection, shading
- **Rasterization Pipeline**
- **Ray-Tracing versus Rasterization**
  - Benefits & Drawback
  - Open Issues

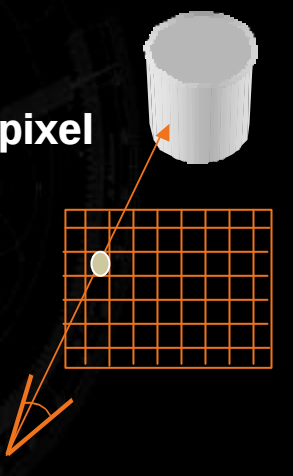
SIGGRAPH 2001

## Ray-Tracing Algorithm




## Ray-Generation

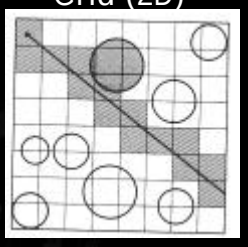
- **Generate initial ray for each pixel**
- **Options**
  - Reuse samples by reprojection (RenderCache)
  - More samples for anti-aliasing
  - Other camera models



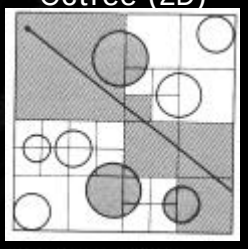
## Ray-Traversal

- **Need to find objects quickly**
- **Build spatial index structure**
  - Grid, octree, BSP-tree, BVH, ...
- **Advantages**
  - Logarithmic complexity
  - Occlusion culling
- **Problems**
  - Multiple intersection computations
  - Dynamic scenes

Grid (2D)



Octree (2D)



SIGGRAPH 2001

## Ray-Object-Intersection

- **Need to compute intersections fast**
  - Requires many floating point operations
  - But dominated by traversal (2:1 – 4:1)
  - Plenty of algorithms
    - Choice depending on input data and environment
- **Optimizations**
  - Use SIMD CPU-extensions (SSE, AltiVec, 3D-Now)
    - Data parallel execution
  - Proper caching of data

SIGGRAPH 2001

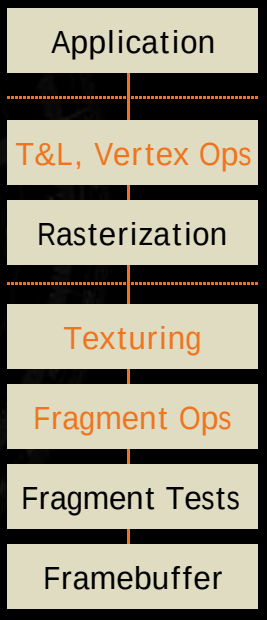
## Shading

- **Shading after visibility has been computed**
  - No overhead due to overdraw
  - Every ray is shaded ***exactly once***
- **Can generate new ray**
  - Shadow, reflection, transmission, ...
    - Need to deal with recursion
- **Direct use of Shading Languages**
  - RenderMan (e.g. BMRT) and others

SIGGRAPH 2001

## Rasterization Pipeline

- **Efficient HW implementation**
  - Use of object coherence
  - Speed (up to 45 Mtris/s)
  - New features
- **Rendering is driven by App.**
  - Application submits geometry
- **Visibility determined at end**
  - Z-buffer fragment test



## Ray-Tracing versus Rasterization

- **Occlusion Culling & Logarithmic Complexity**
  - RT never even looks at invisible geometry
  - RT traversal allows for efficient searching O(log N)
  - Rasterization shows linear behavior
  - RT wins for complex scenes
  - However: Rasterization can be improved
    - Early Z-buffer test (e.g. ATI´s Hyper-Z)
    - HW-assisted occlusion test
      - – Requires similar index structure

## Ray-Tracing versus Rasterization

- **Flexibility**
  - Handling individual or unstructured groups of rays
    - Image-based rendering & RenderCache
- **Correctness & Image Quality**
  - Rasterization relies on approximations
    - Environment maps, shadow maps, …
  - Ray-Traced images are “correct” by default
    - Shadows, reflections, refractions, …
    - Use of approximations is optional

## Ray-Tracing versus Rasterization

- **Simple and Efficient Shading**
  - No overhead
  - Direct use of Shading Languages
- **Parallel Scalability**
  - Ray-Tracing is „embarrassingly parallel“
  - Should scale well with hardware
  - Initial hardware cost is higher than for rasterization


## Ray-Tracing versus Rasterization

- **Coherence**
  - Key to efficient rendering
  - Rasterization: Object coherence
    - Efficient rasterization
  - Ray-Tracing: Ray coherence
    - Improved caching & reduced bandwidth
    - Allows for data parallel computation
  - RT has much more coherence than assumed

## Open Research Problems

- **Hardware**
  - What is the best HW architecture?
- **Dynamic Scenes**
  - Optimized rebuild or transformation of index?
- **API**
  - Better alternative to OpenGL´s „push model“?
- **Can RT eventually replace rasterization?**